

FIG.1

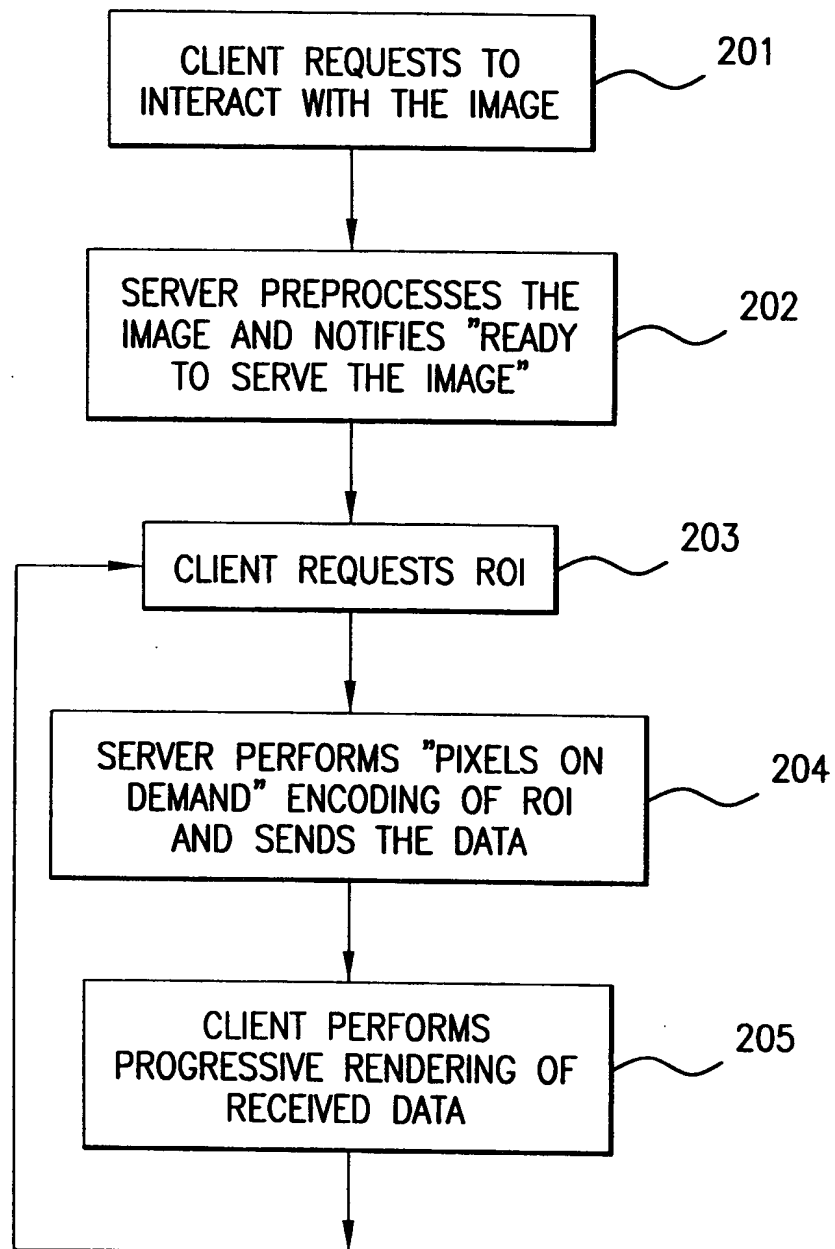


FIG.2

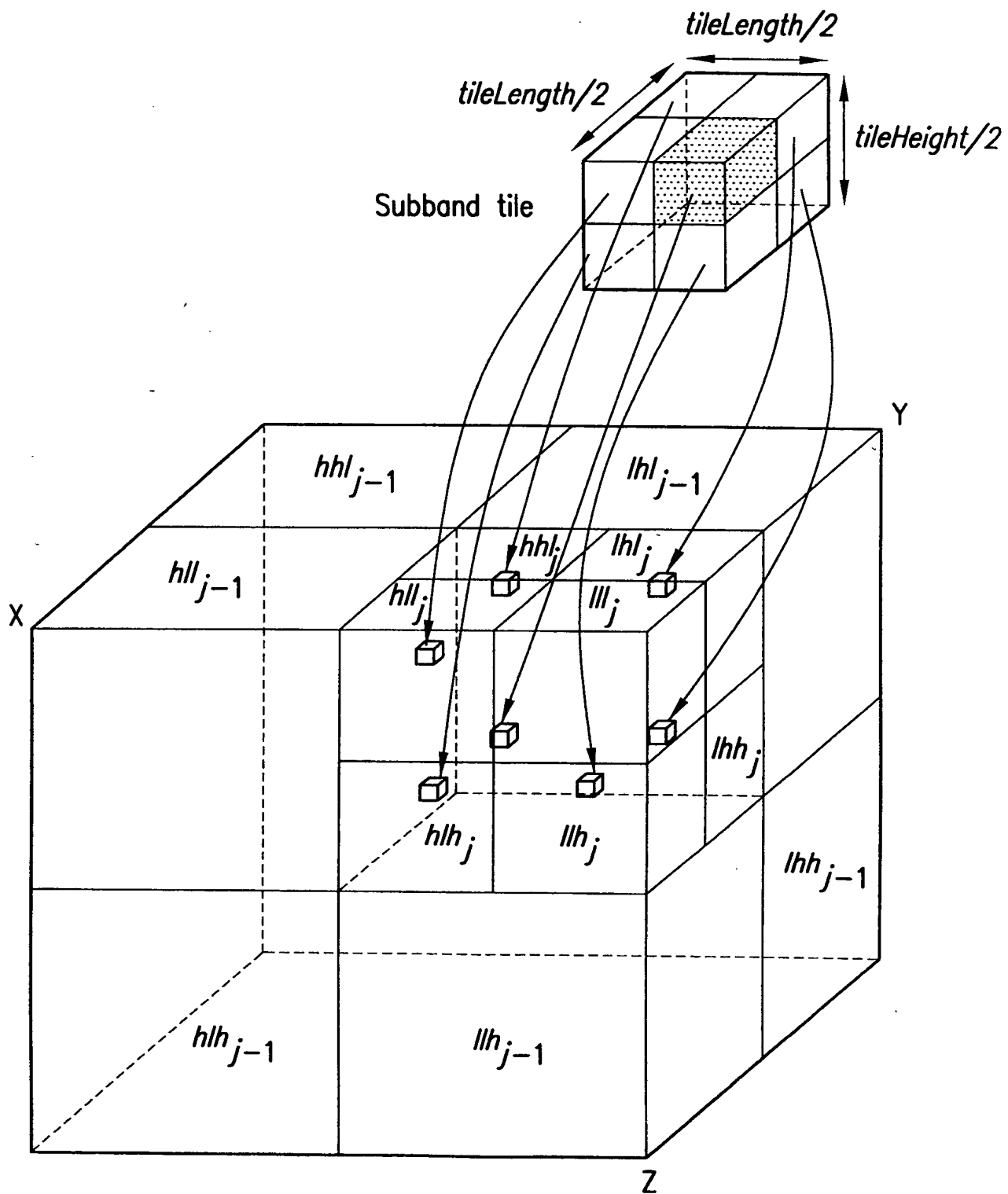


FIG.3

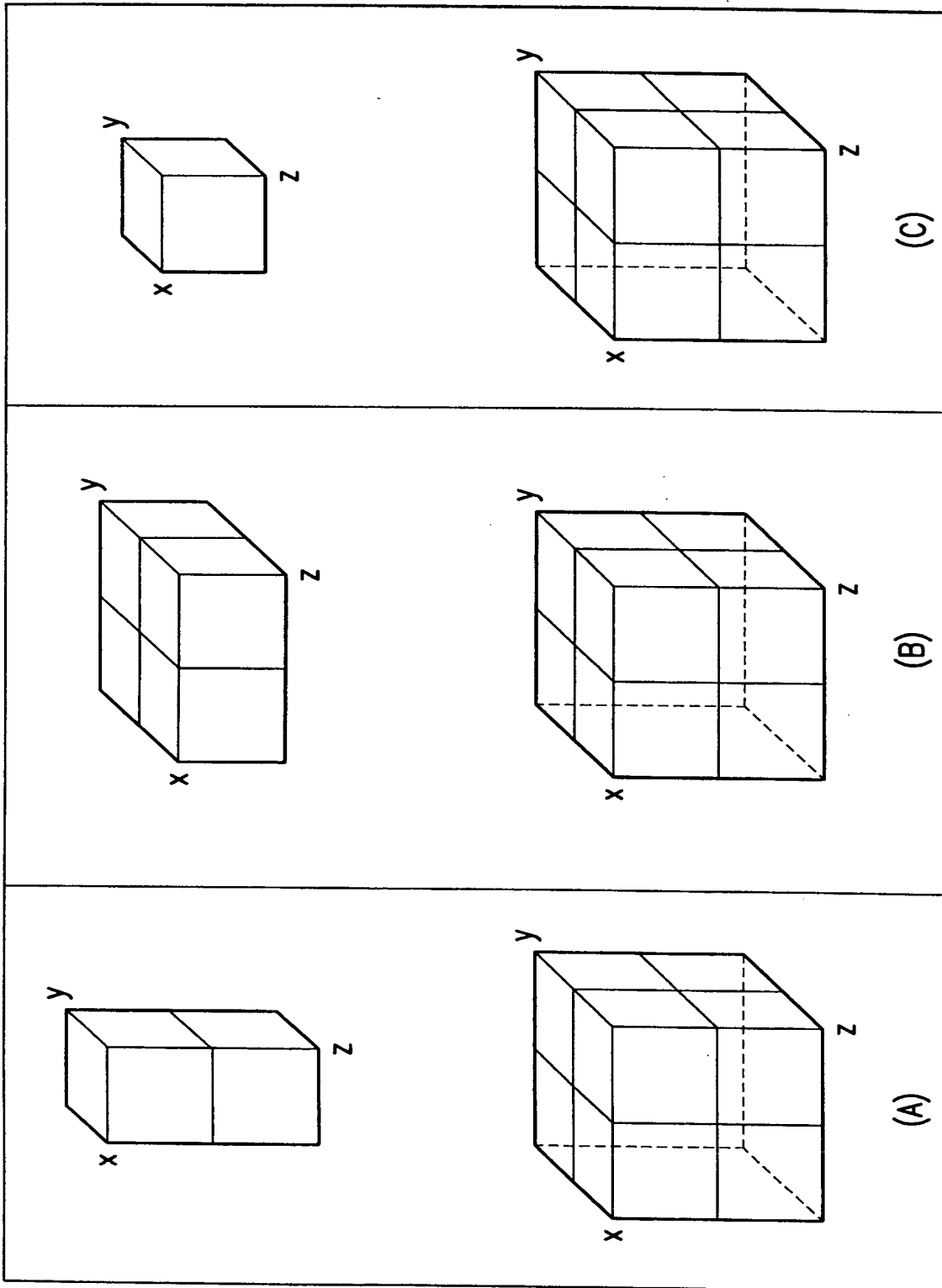


FIG.4

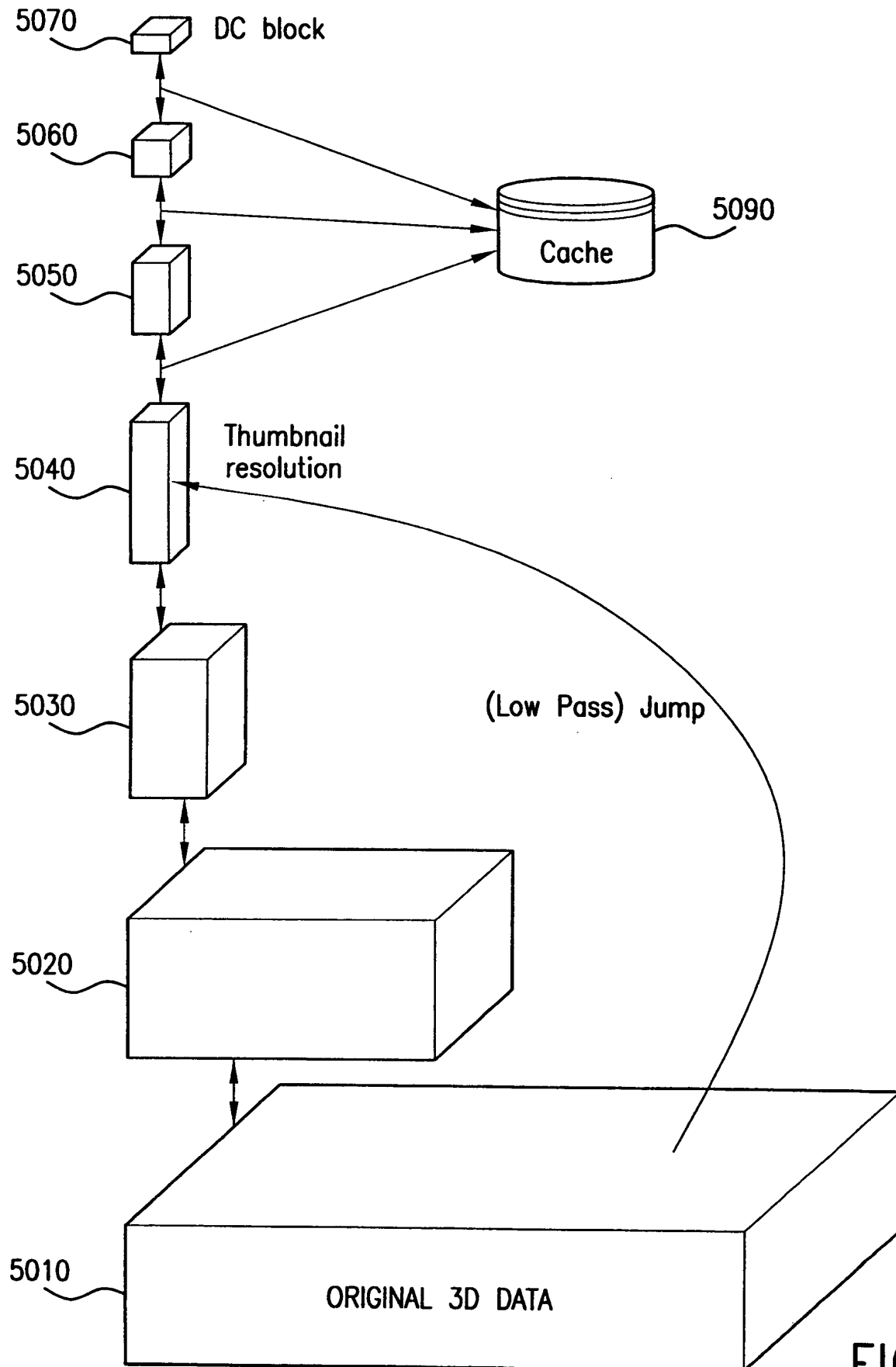
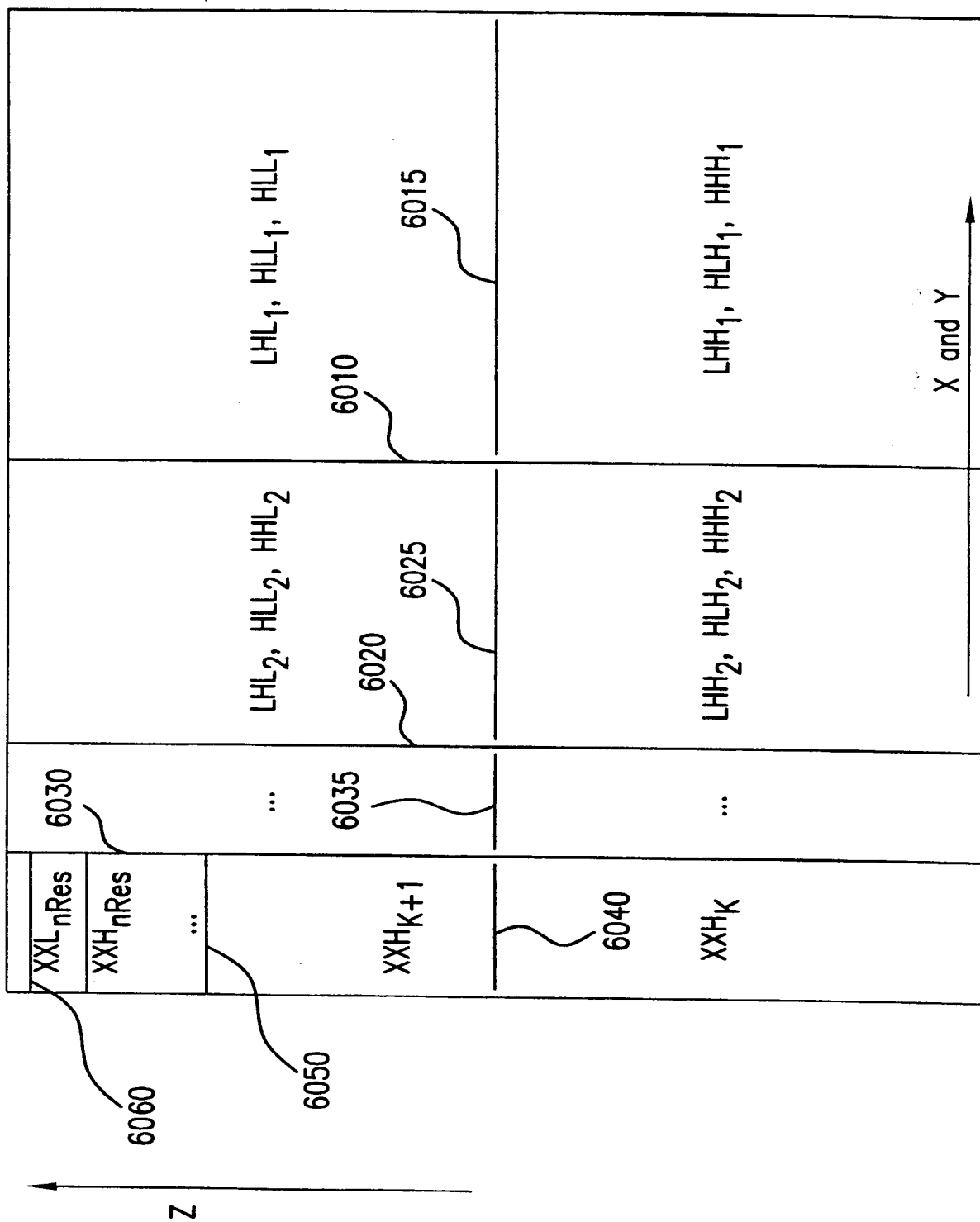


FIG.5



**FIG. 6**

z	...	...			$\sqrt{2}$
	1	$\sqrt{2}$			$\sqrt{2}$
	$\sqrt{2}$	$\sqrt{2}$			$\sqrt{2}$
	1	$\sqrt{2}$			$\sqrt{2}$

FIG.7

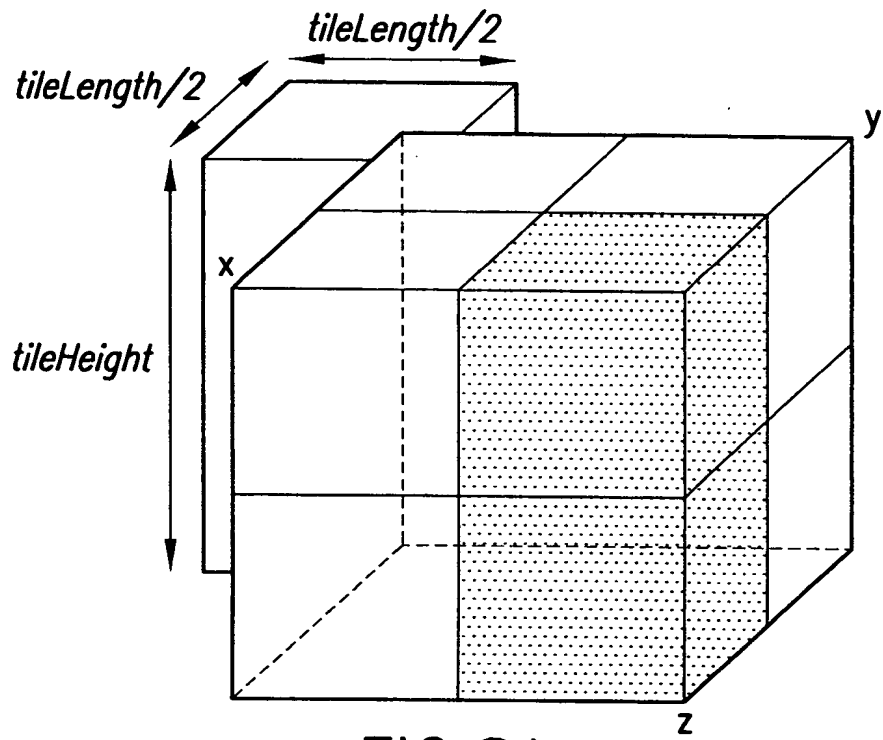


FIG.8A

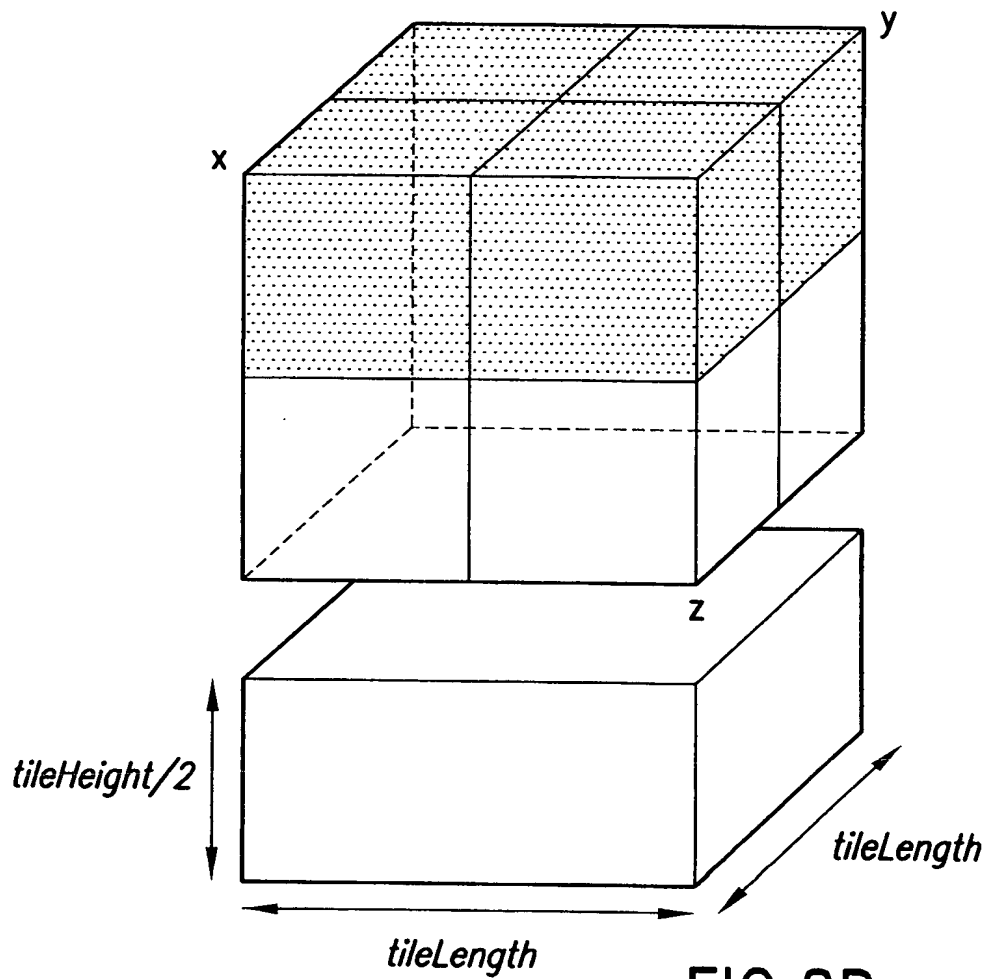


FIG.8B



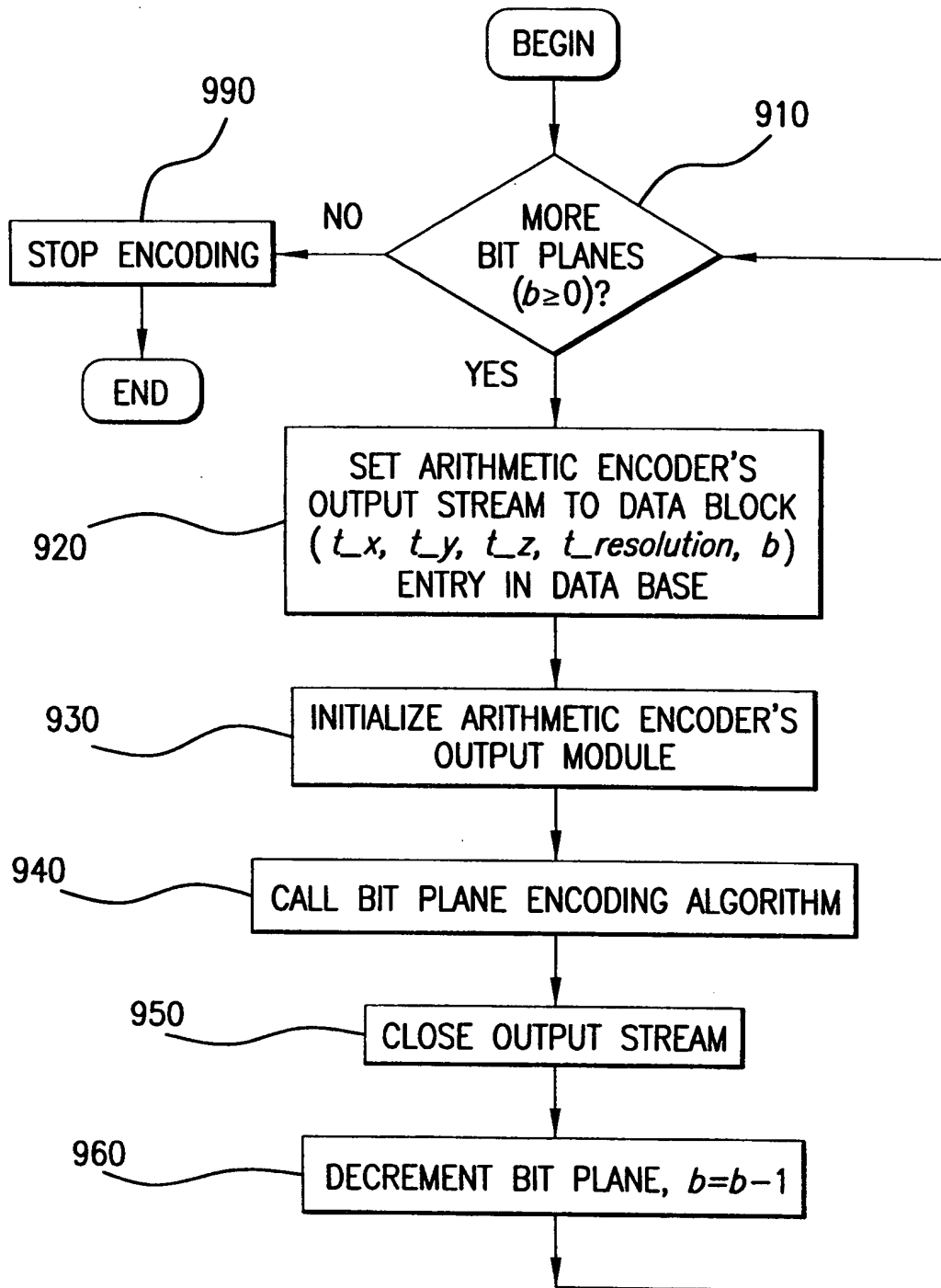


FIG.9

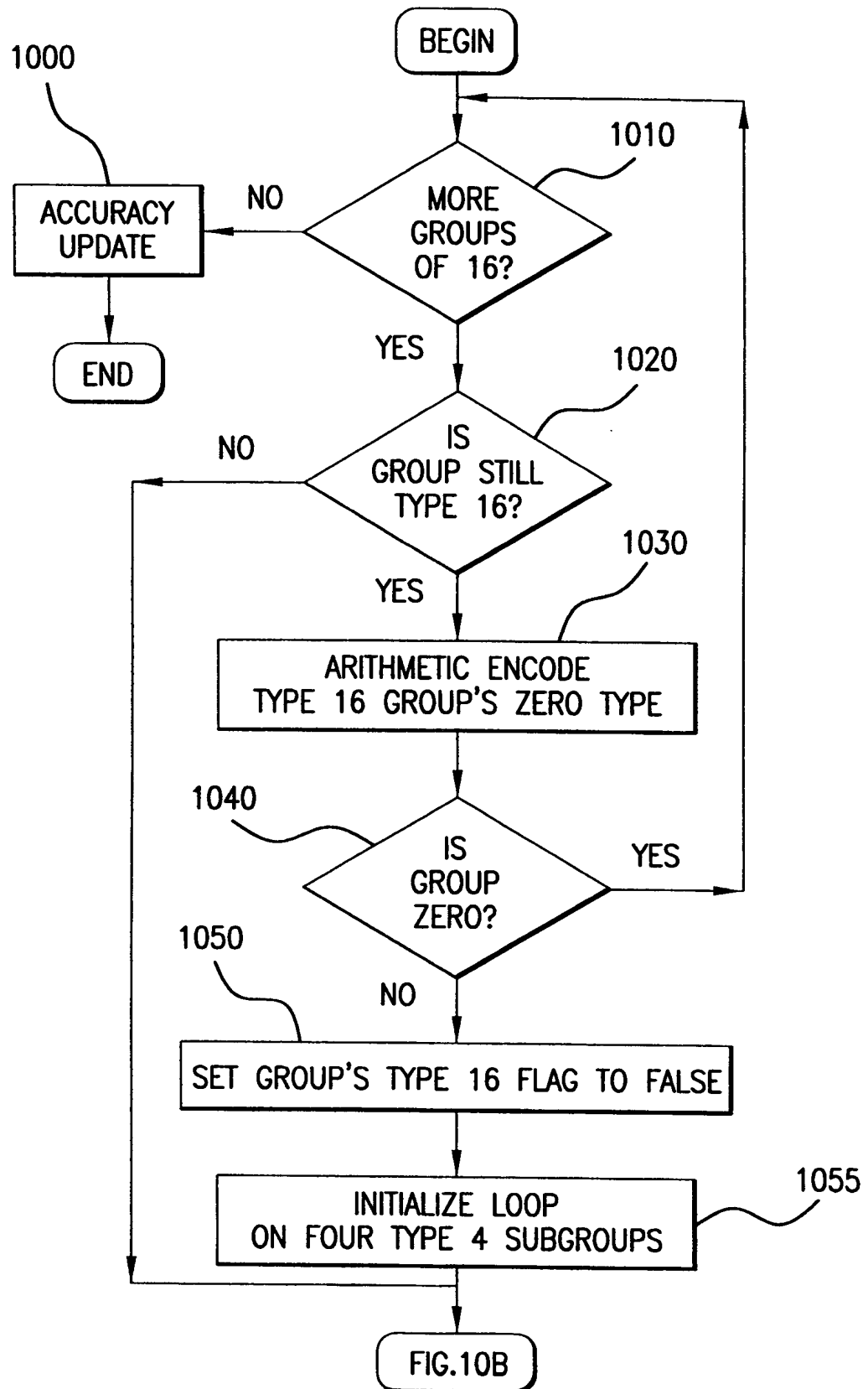


FIG.10A

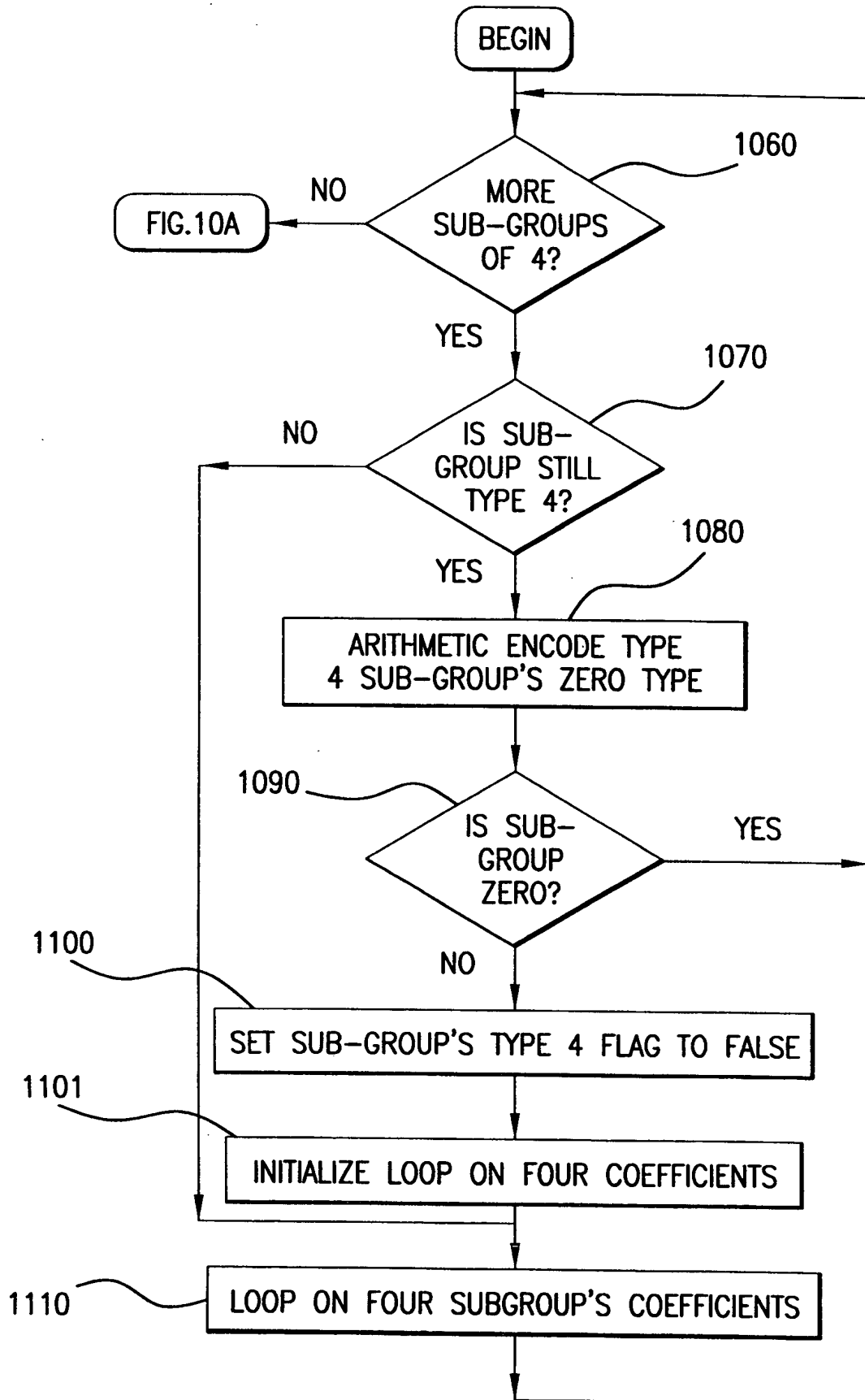


FIG. 10B

```

zeroModel_16.start_model();
zeroModel_4.start_model();
zeroCoefModel.start_model();
coefSignModel.start_model();

while(encoder.getNextGroupOf16()) {
    bool isZero;

    if (encoder.isGroupType16()) {
        isZero = encoder.isZeroGroupOf16();
        arithmetic_encode_symbol(ZeroModel_16,isZero);
        if (isZero)
            continue;
    }

    while (encoder.getNextGroupOf4()) {
        if (encoder.isGroupType4()) {
            if (!encoder.mustBeNoZeroGroup()) {
                isZero = encoder.isZeroGroupOf4();
                arithmetic_encode_symbol(ZeroModel_4,isZero);
                if (isZero)
                    continue;
            }
        }

        while (encoder.getNext_Type1_Coef(isZero)) {
            if (!encoder.mustBeNoZeroCoef())
                arithmetic_encode_symbol(zeroCoefModel,isZero);
            if (!isZero)
                arithmetic_encode_symbol(coefSignModel,encoder.getCoefSign());
        }
    }

    if (!(encoder.isLastBitPlane() && equalBinSetting)) {
        bitModel.start_model();

        int bit;
    }
}

```

FIG. 11

```
bitModel.startModel();
zeroCoefModel.startModel();
coefSignModel.startModel();
while (encoder.moreCoef()) { ~ 1210
    if (encoder.isCoefReported()) { ~ 1220
        arithmetic_encode_symbol(
            bitModel, encoder.reportedCoefPrecisionBit());
    } else {
        if ( encoder.isCoefExactZero() ) ~ 1230
            arithmetic_encode_symbol(zeroCoefModel, true);
        else {
            arithmetic_encode_symbol(zeroCoefModel, false);
            arithmetic_encode_symbol(
                coefSignModel, encoder.getCoefSign());
        }
    }
}
```

FIG. 12A

```
bitModel.startModel();  
  
for (int z = 0 ; z != HalfBitPlaneZSize;z++) {  
    for (int y = 0 ; y != HalfBitPlaneYSize;y++) {  
        for (int x = 0 ; x != HalfBitPlaneXSize;x++) {  
            arithmetic_encode_symbol(bitModel,coefHalfBit[x][y][z]);  
        }  
    }  
}
```

FIG. 12B

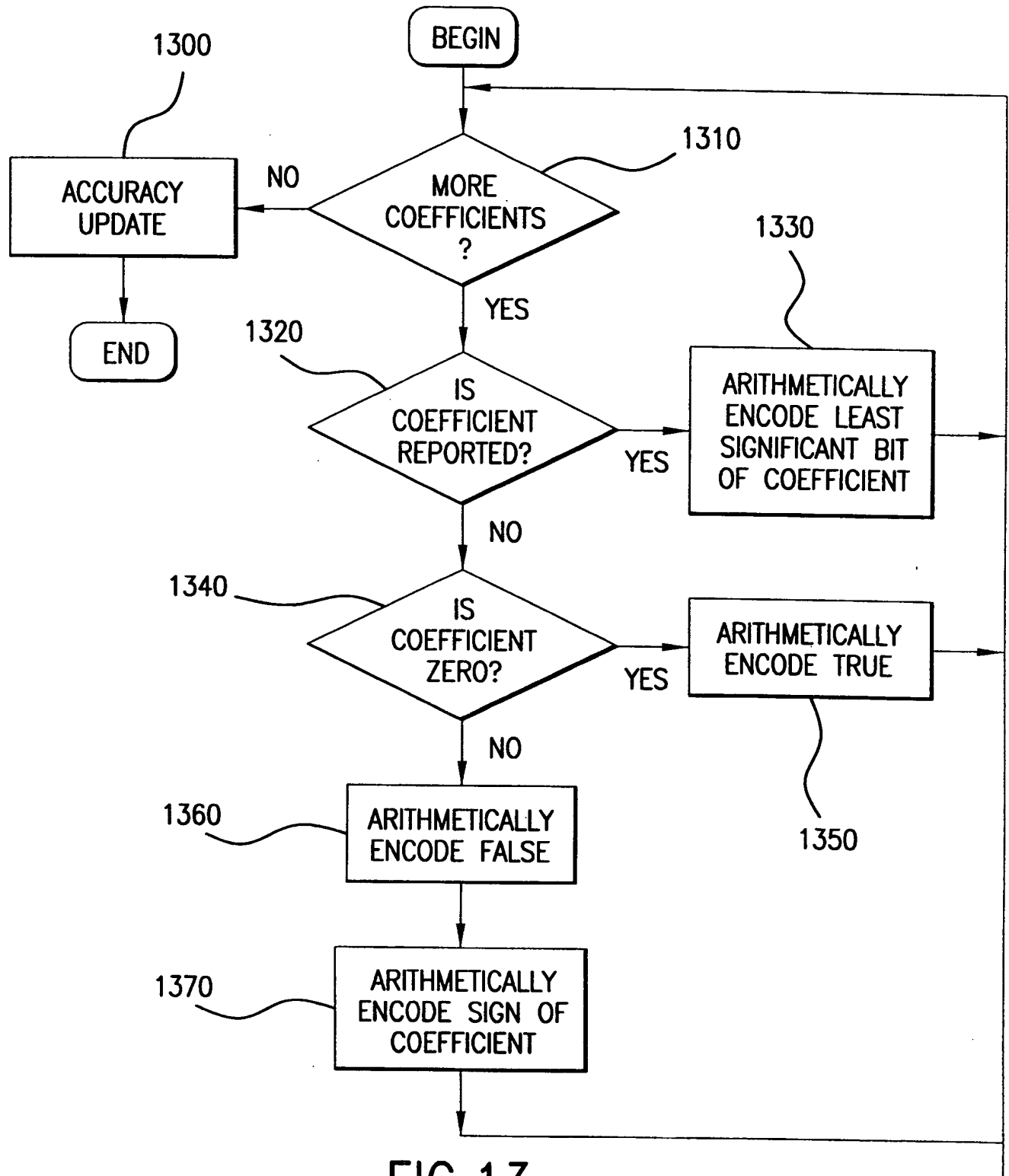


FIG.13

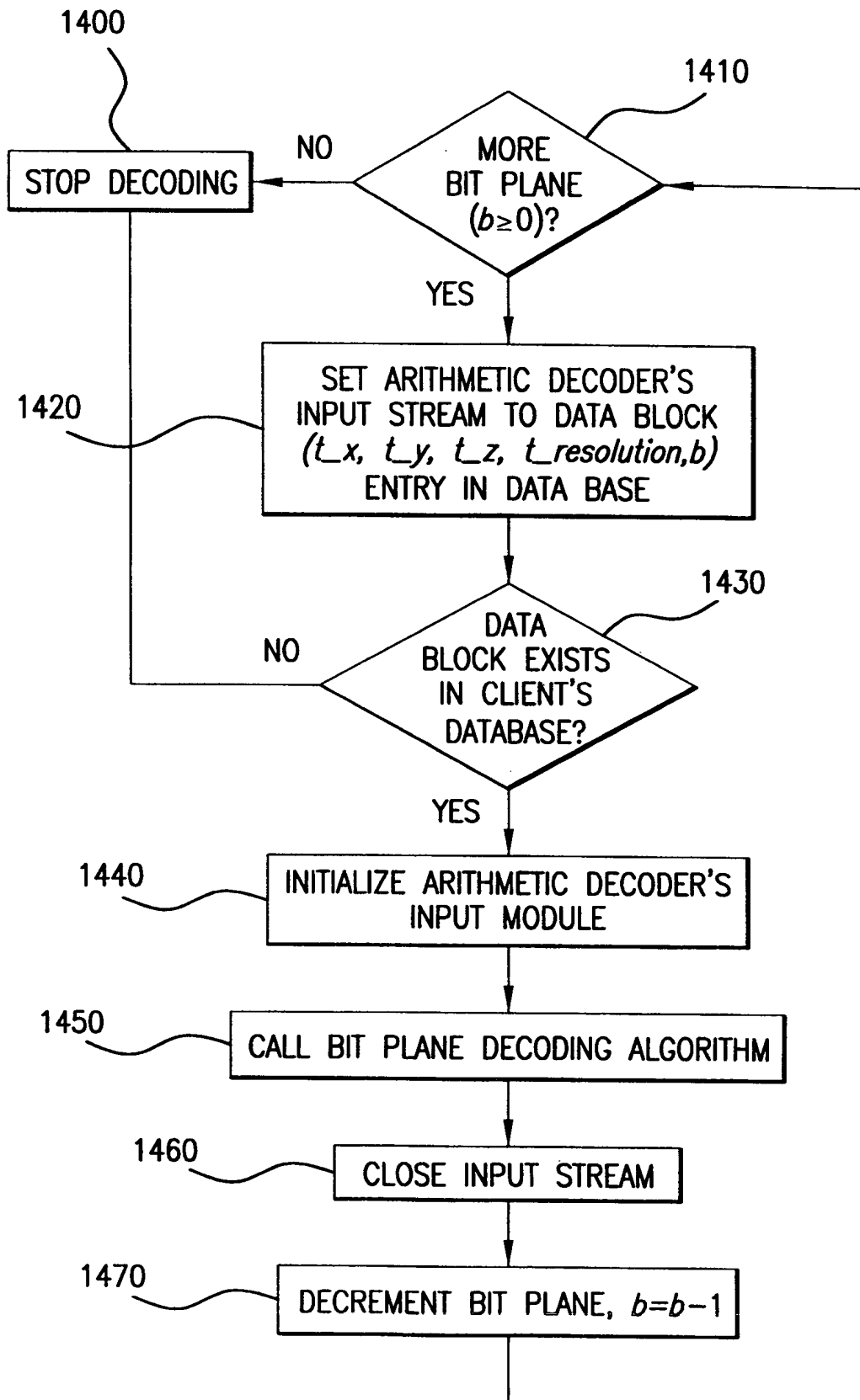


FIG.14



```

zeroModel_16.start_model();
zeroModel_4.start_model();
zeroCoefModel.start_model();
coefSignModel.start_model();

while(decoder.getNextGroupOf16()) {
    if (decoder.isGroupType16()) {
        if (arithmetic_decode_symbol(zeroModel_16)) {
            decoder.zeroGroupOf16();
            continue;
        }
        else
            decoder.removeZeroGroupOf16();
    }

    while (decoder.getNextGroupOf4()) {
        if (decoder.isGroupType4()) {
            if (!decoder.mustBeNotZeroGroup()) {
                if (arithmetic_decode_symbol(zeroModel_4)) {
                    decoder.zeroGroupOf4();
                    continue;
                }
            }
            decoder.removeZeroGroupOf4();
        }

        while (decoder.getNext_Type1_Coef()) {
            if (decoder.mustBeNotZeroCoef())
                decoder.setNextSigCoef(arithmetic_decode_symbol(coefSignmodel));
            else if (!arithmetic_decode_symbol(zeroCoefModel))
                decoder.setNextSigCoef(arithmetic_decode_symbol(coefSignmodel));
        }
    }
}

if (! (decoder.isLastBitPlane() && equalBinSetting)) {
    bitModel.start_model();
    while(decoder.moreSignificantCoef())
        decoder.setSignificantCoefBit(arithmetic_decode_symbol(bitModel));
}

```

FIG. 15

```

bitModel.startModel();
zeroCoefModel.startmodel();
coefSignModel.startmodel();

decoder.initializeLSBPlaneCoefScan();
while (decoder.moreCoef()) {
    if (decoder.isCoefReported()) {
        if (decoder.isSkippedCoef()) {
            decoder.updateLSB (0);
        }
        else {
            decoder.updateLSB(arithmetic_decoder_symbol(bitModel));
        }
    }
    else {
        if (!decoder.isSkippedCoef()) {
            if (!arithmetic_decoder_symbol(zeroCoefModel))
                decoder.setLSB(arithmetic_decoder_symbol(coefSignModel));
        }
    }
}

```

1610

FIG. 16A

```

bitModel.startModel();
for (int z = 0 ; z != HalfBitPlaneZSize;z++) {
    for (int y = 0 ; y != HalfBitPlaneYSize;y++) {
        for (int x = 0 ; x != HalfBitPlaneXSize;x++) {
            coefHalfBit[x][y][z] = arithmetic_decoder_symbol(bitModel);
        }
    }
}

```

FIG. 16B

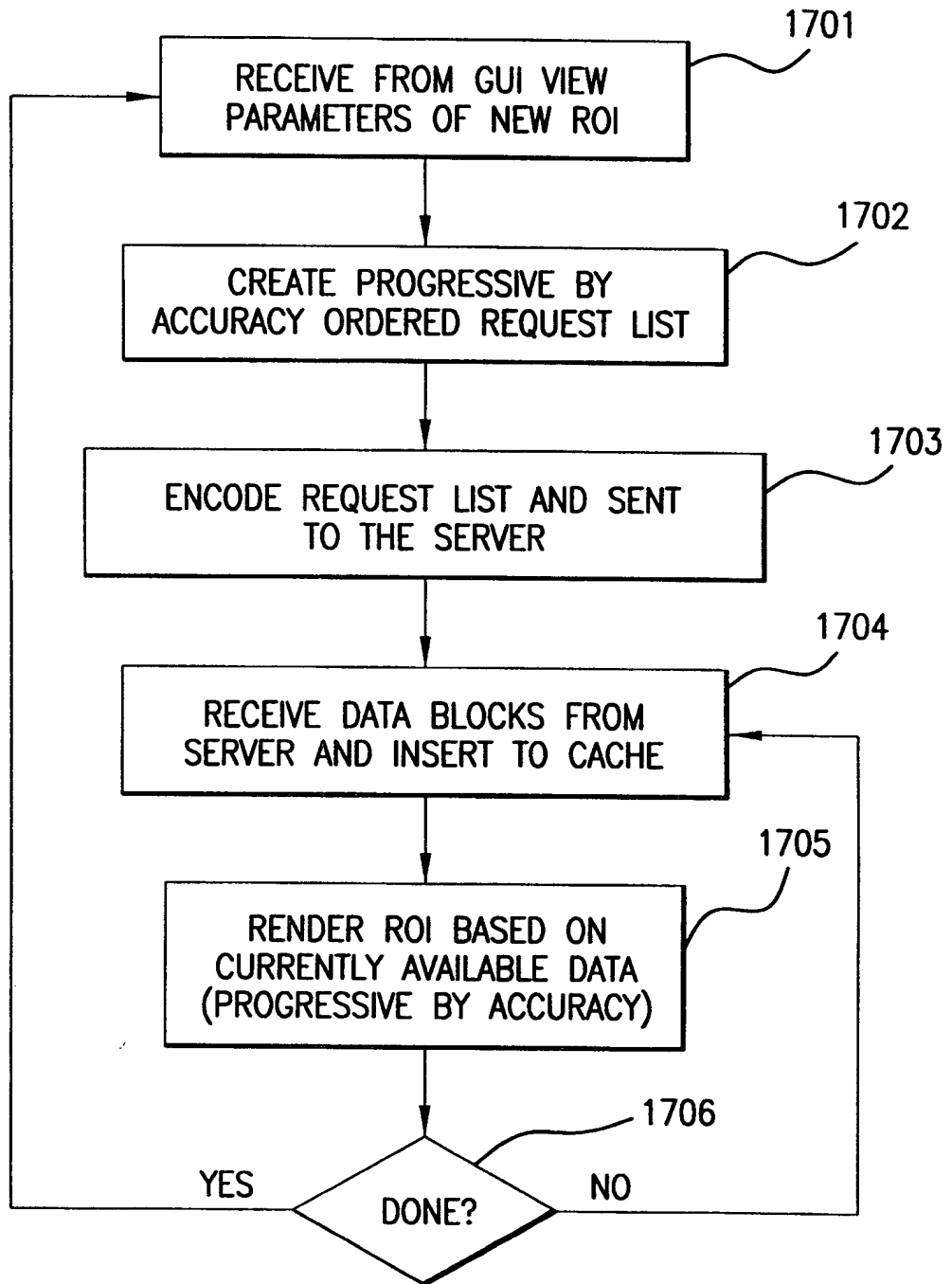


FIG.17

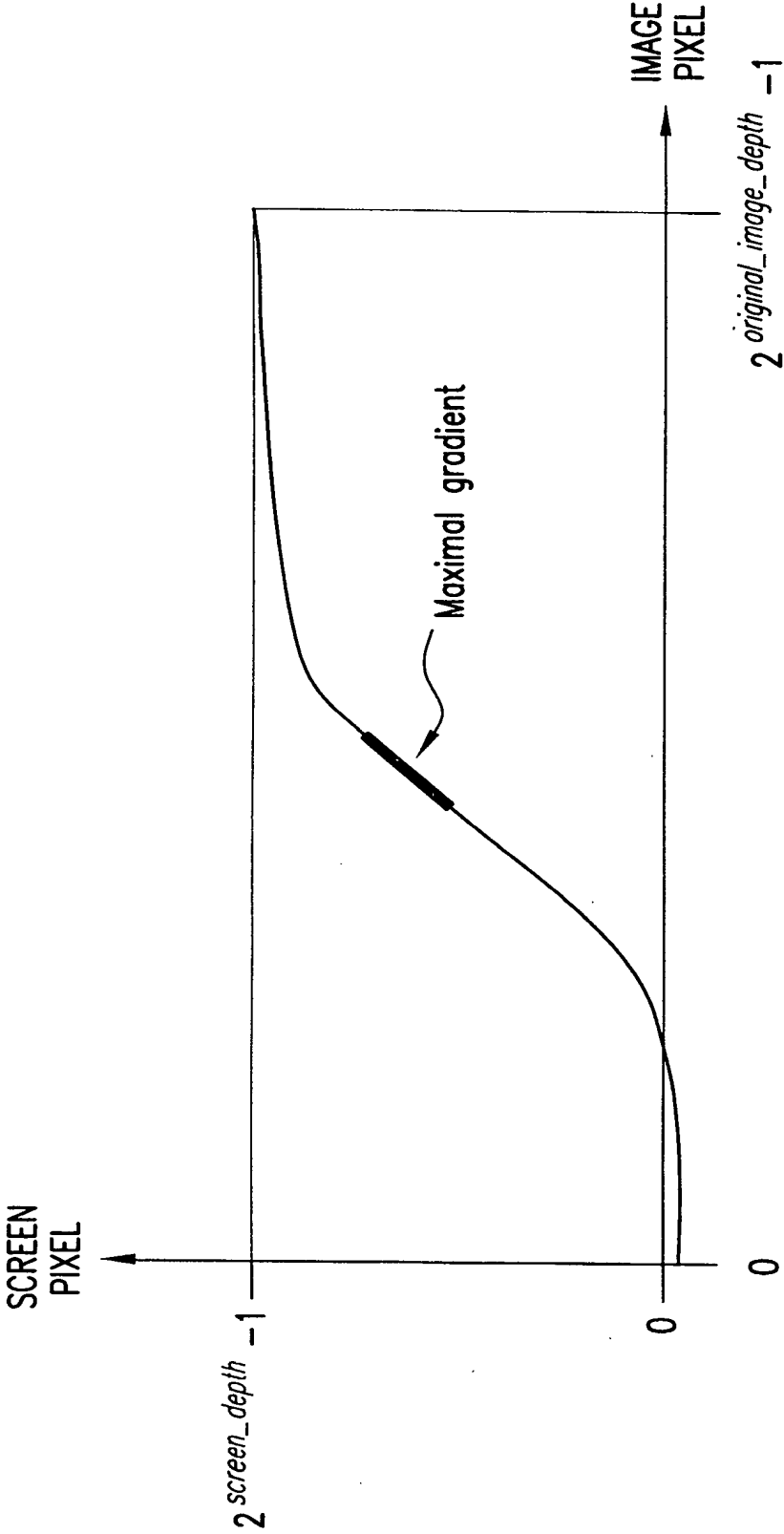


FIG.18

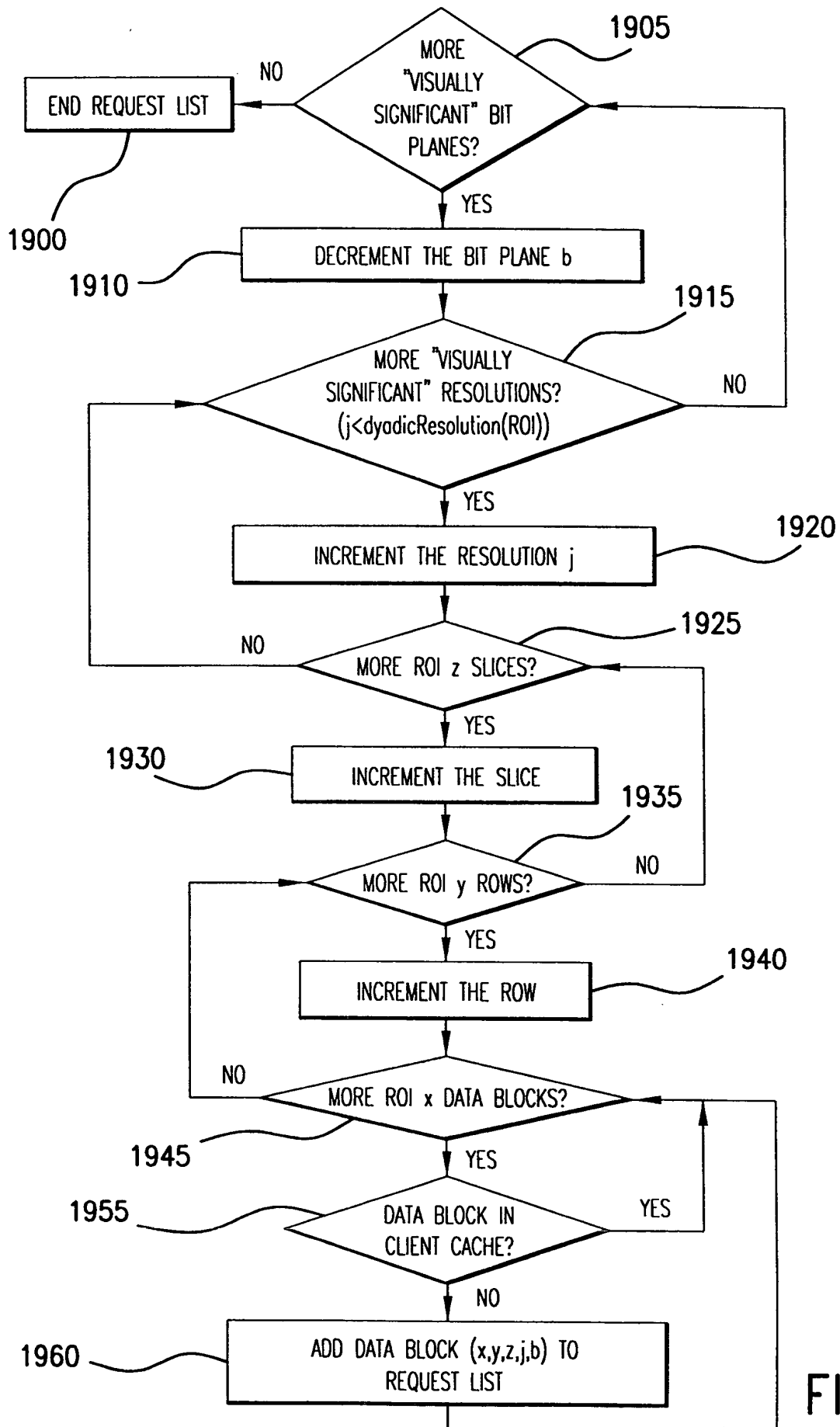


FIG.19

```

for (int res = 1 ; res <= dyadicResolution(ROI); res++) {
    for( int z=0;
        z < NumberOfZtilesOnDyadicResolution (res,ROI);
        z++ ) {

        GetCoefficientsOfLowerResolution(res, Ztile);

        for( int x=0;
            x < NumberOfXtilesOnDyadicResolution(res,ROI);
            x++ ) {
            for( int y=0;
                y <
                NumberOfYtilesOnDyadicResolution(res,ROI);
                y++ ) {
                DecodeOrExtractFromCacheSubbandCoefficients
                ( res, x, y, z );
            }
        }

        ExecuteInverseSubbandTransform(z);

        if( res == dyadicResolution(ROI))
            ImageResizeAndMappingTo8bitScreen();
    }
}

```

FIG. 20

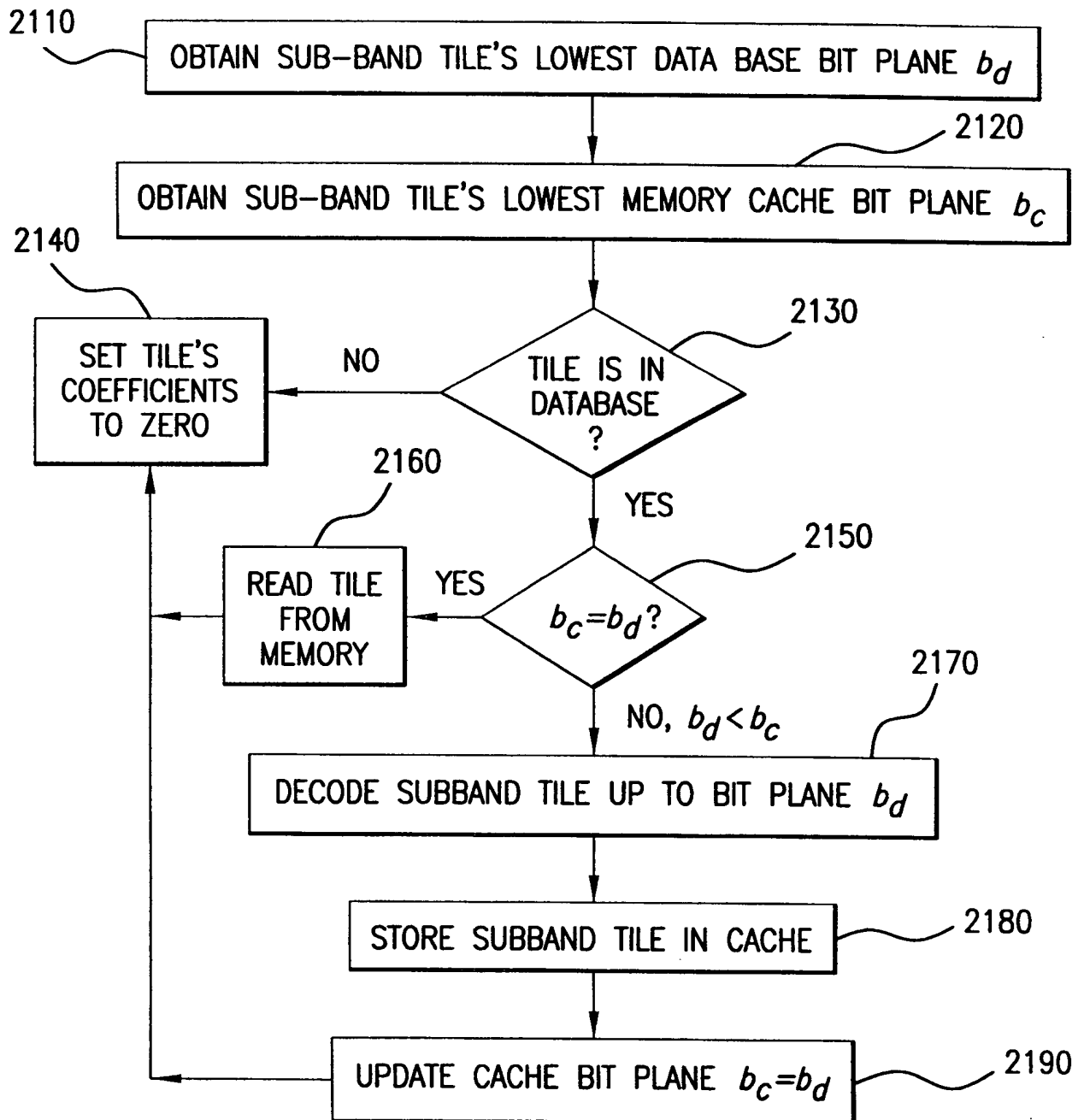


FIG.21

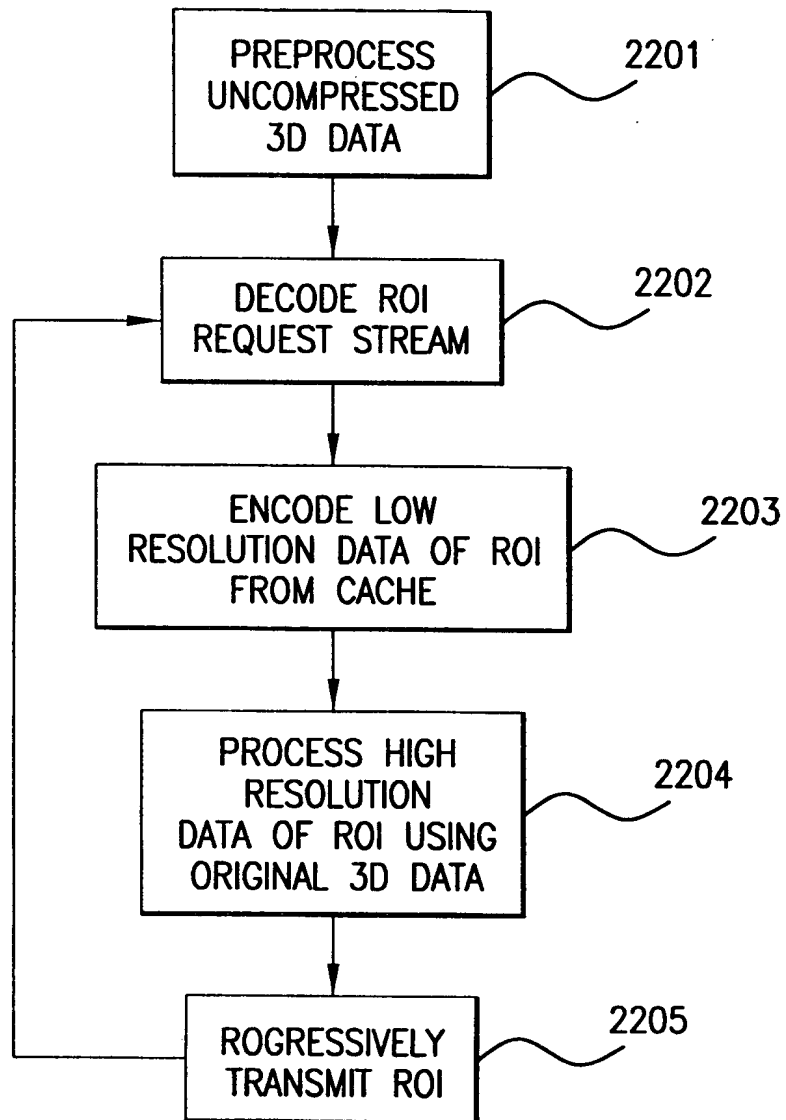


FIG.22



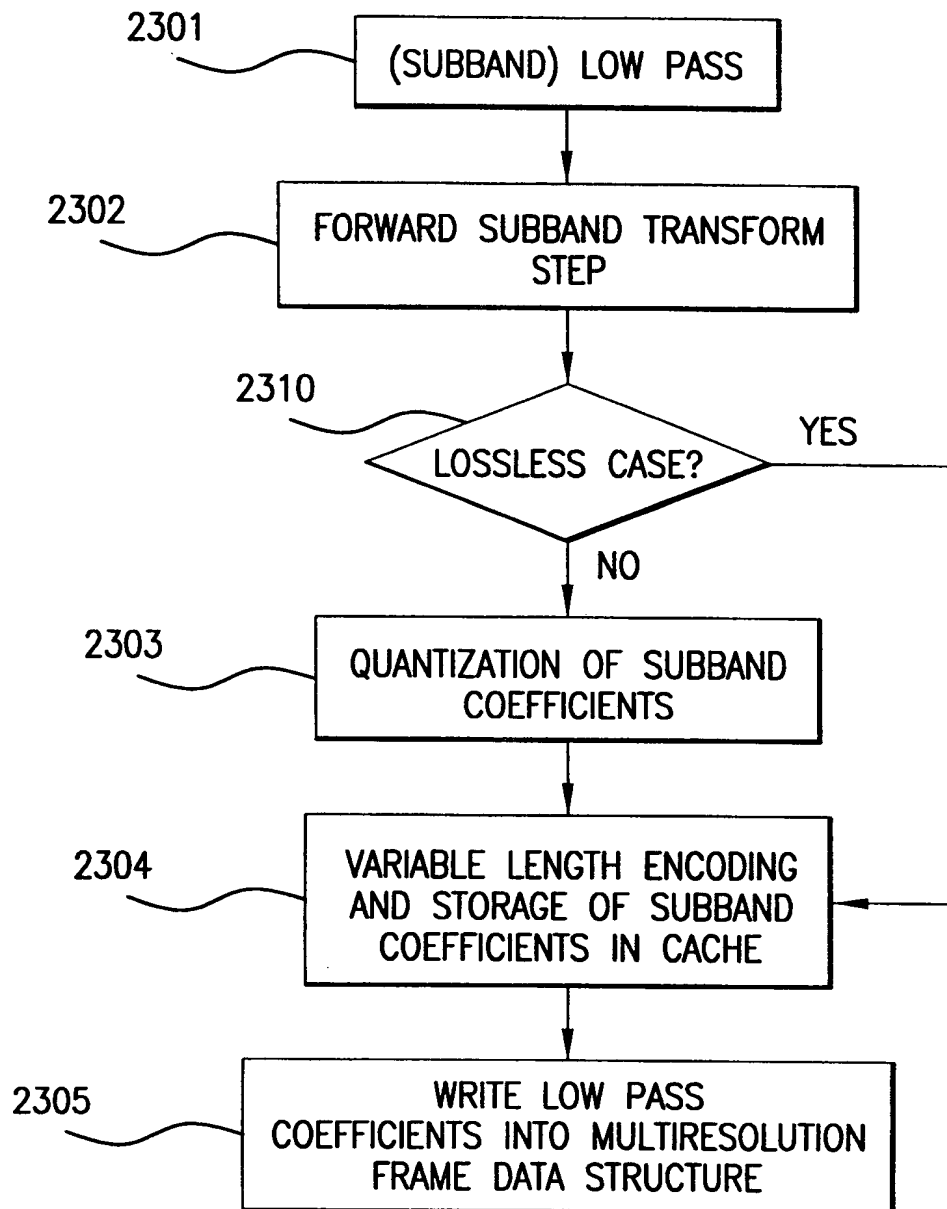


FIG.23

```

for(int t_Resolution=numberOfResolutions-jumpSize; t_Resolution>=1 ;
t_Resolution--) {
    leftTilesZInMemoryBuffer(t_Resolution)=
        NumberOfTilesZInFrameMemoryBuffer(t_Resolution);
    currentTile(t_Resolution)=0;
}

for(t_Resolution=numberOfResolutions-jumpSize; ;) {
    // calculate the Z and it's resolution
    if (currentTile(t_Resolution)< nTileZ(t_Resolution)) {
        for (int t_y = 0 ; t_y < nTileY(t_Resolution); t_y++)
            for (int t_x = 0 ; t_x < nTileX(t_Resolution); t_x++)
                preprocessSubbandTile(t_x,t_y,
currentTile(t_Resolution), t_Resolution);
    }

    // update the indices
    leftTilesZInMemoryBuffer(t_Resolution)--;
    currentTile(t_Resolution)++;

    if(currentTile(t_Resolution) < nTileZ(t_Resolution)) {
        // switch the resolution
        if(leftTilesZInMemoryBuffer(t_Resolution)==0) {
            leftTilesZInMemoryBuffer(t_Resolution) =
                NumberOfTilesZInFrameMemoryBuffer(t_Resolution
);
            t_Resolution --;
        }
        else
            t_Resolution = numberOfResolutions-jumpSize;
    }
    else {
        t_Resolution --;
        if (t_Resolution < 1)

```

FIG. 24

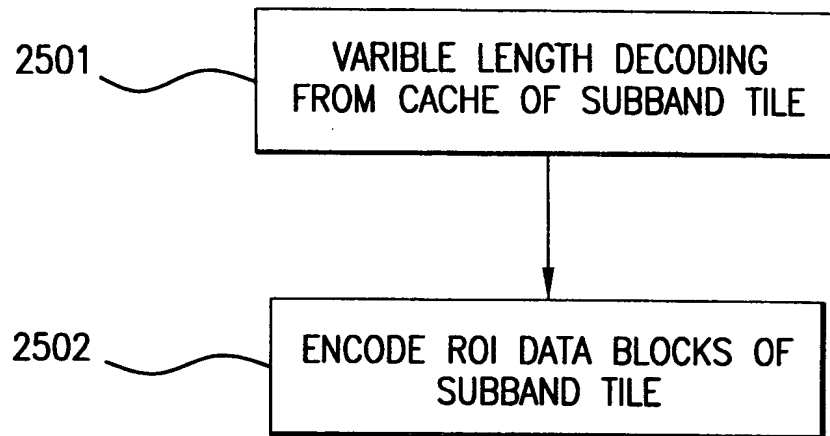


FIG.25

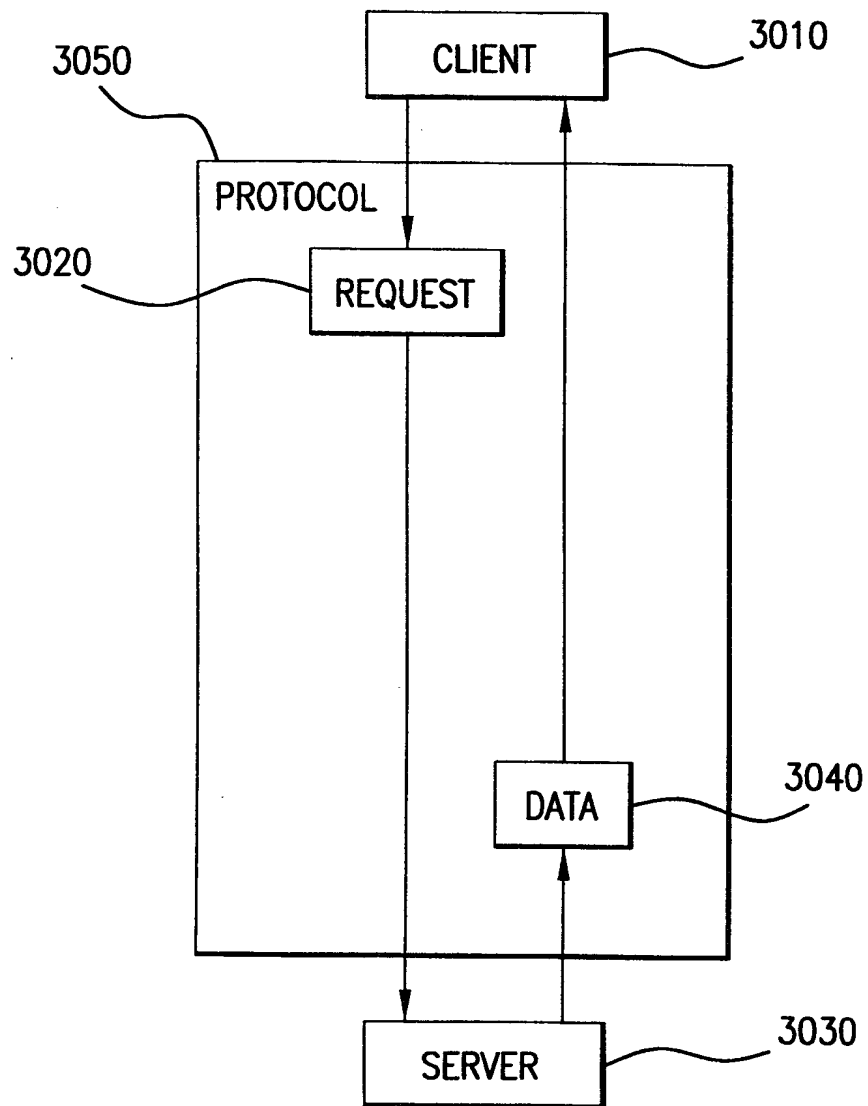


FIG.26

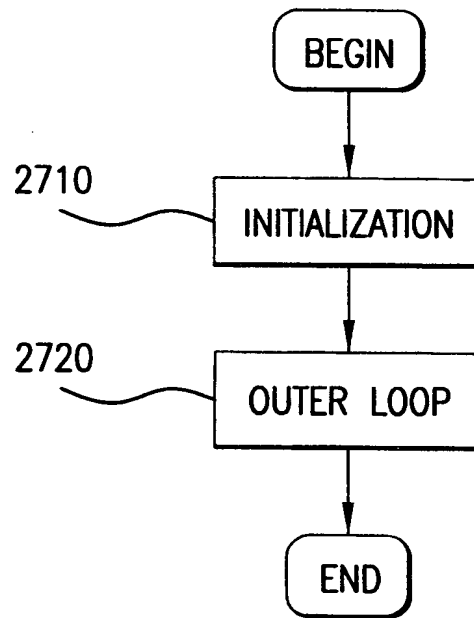


FIG.27

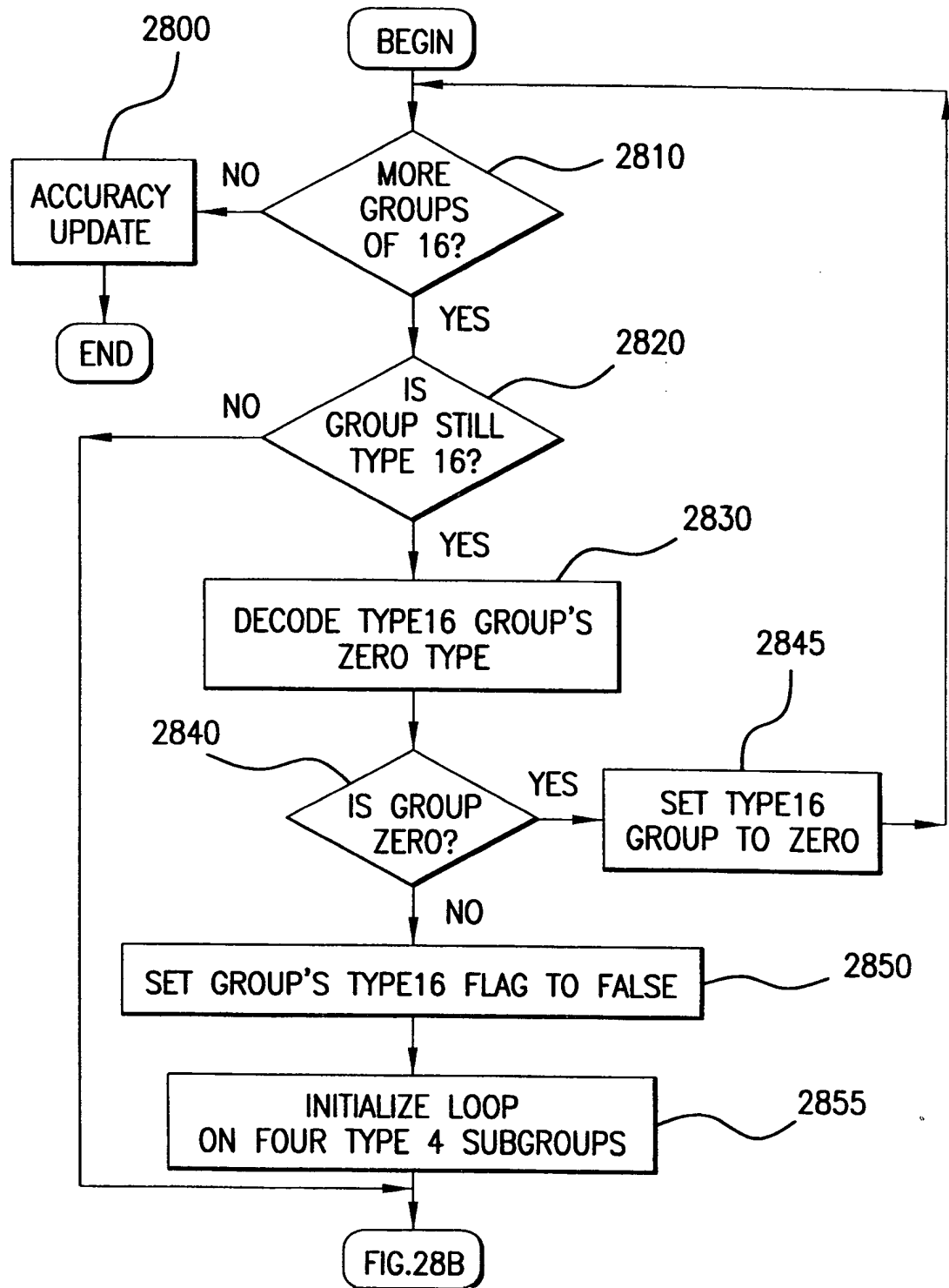


FIG.28A

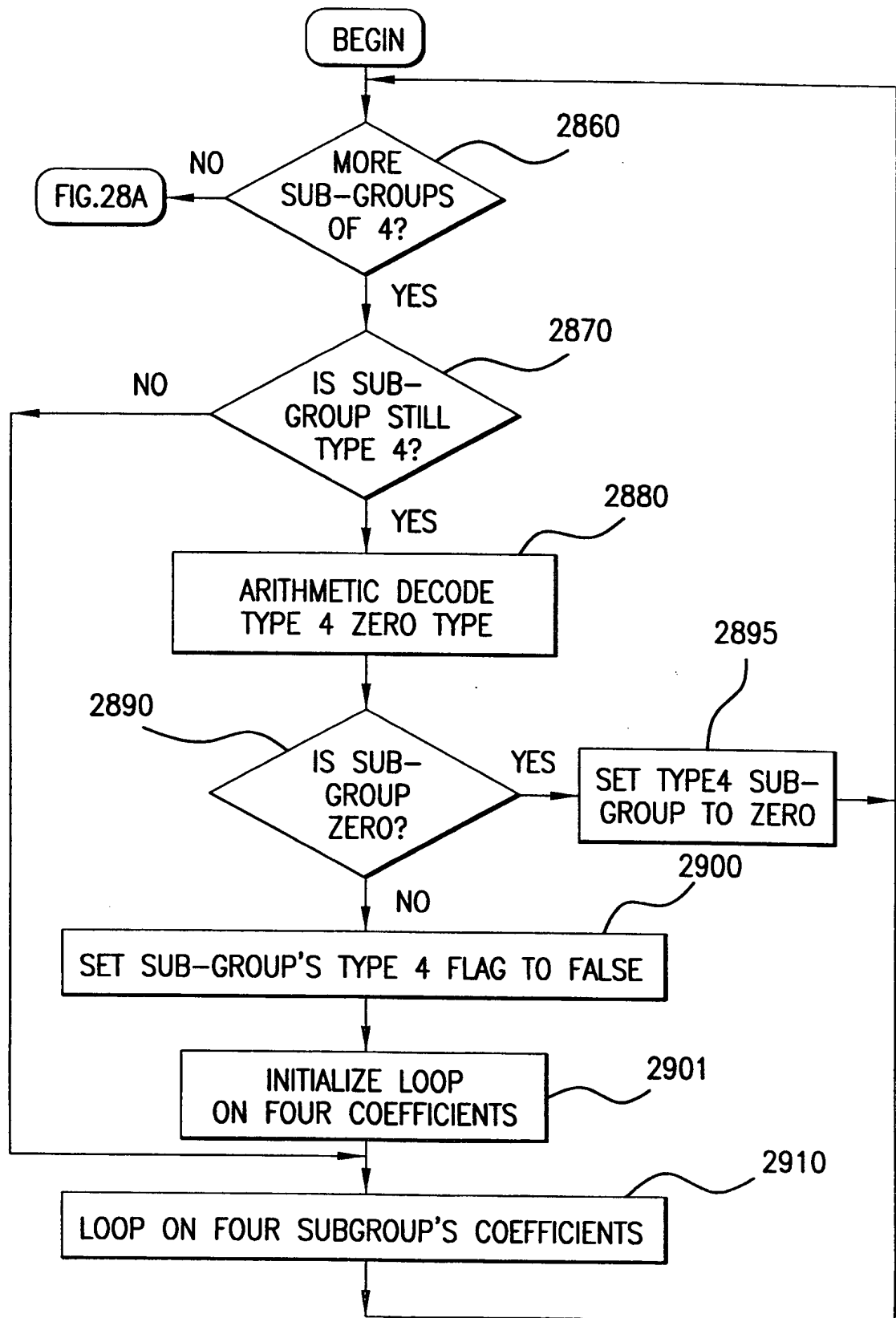


FIG.28B

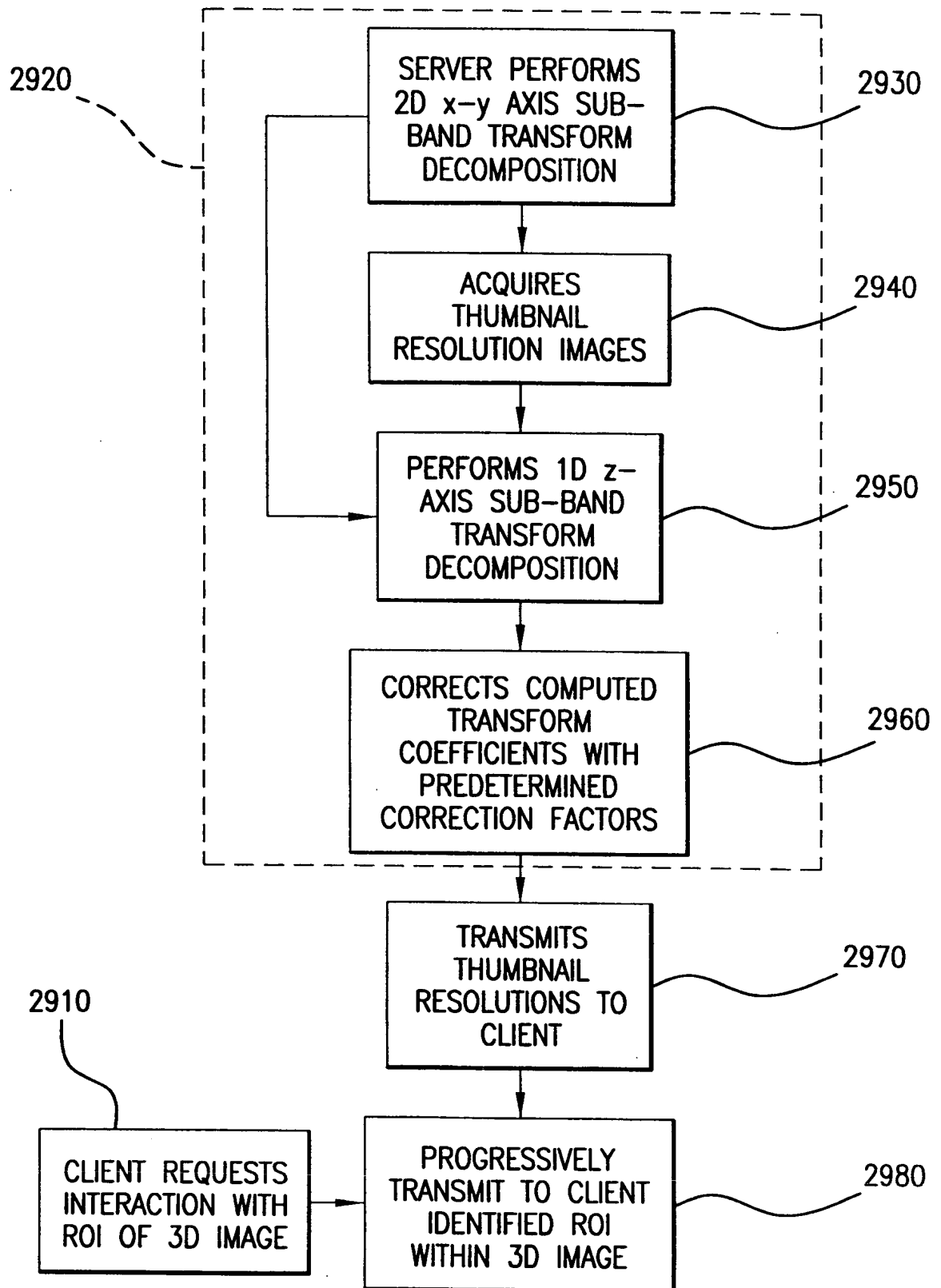


FIG.29



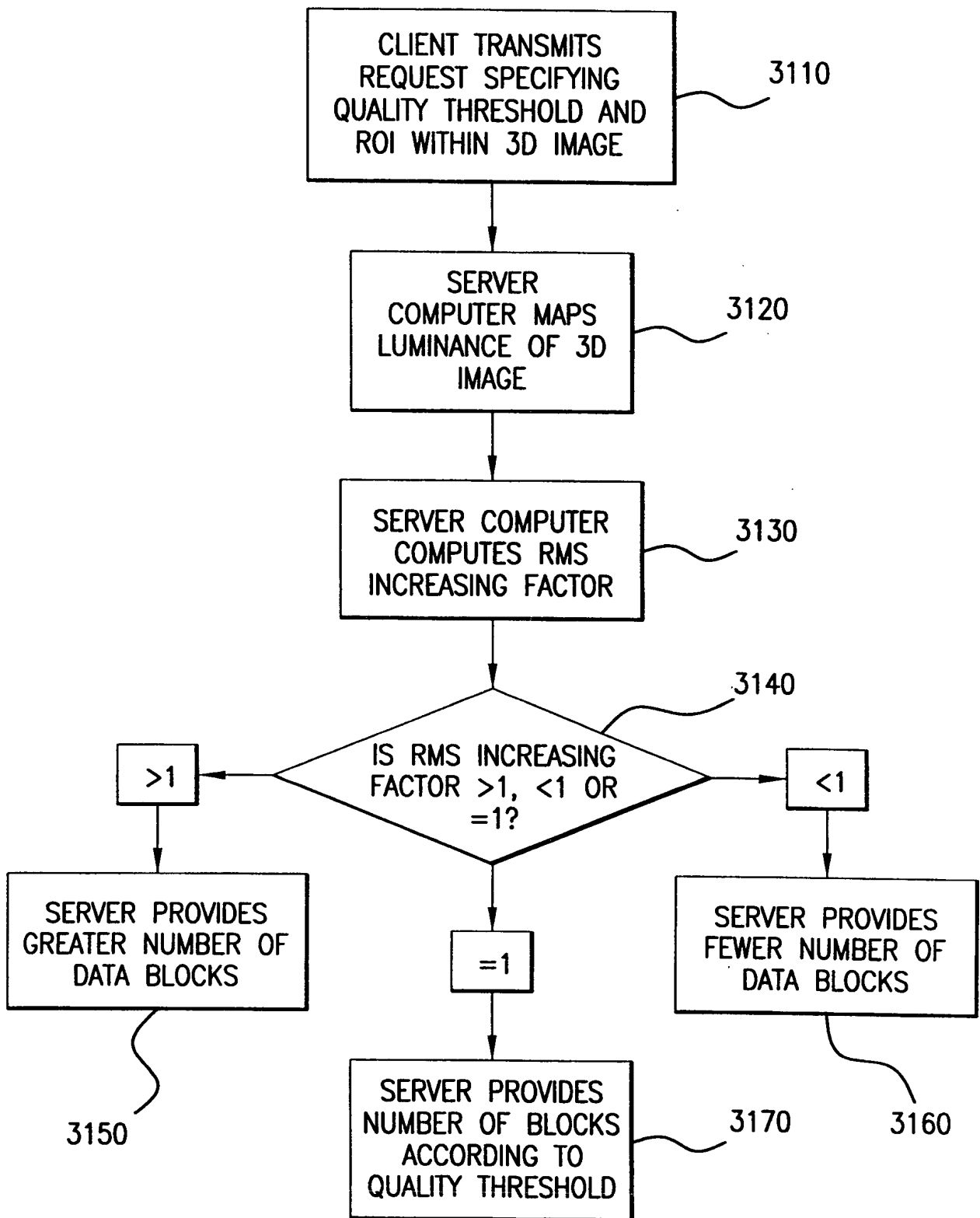


FIG.30